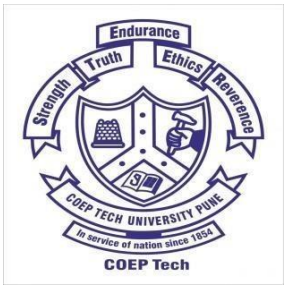


TerrainRover AI: Smart Remote-Controlled Car with Terrain-Aware Navigation

A MINI PROJECT REPORT

Submitted by

Kedar Shiralkar (112107065)
Atharva Thakur (112107070)
Ujjwal Mahajan (112107071)
Pankaj Waingade (112107077)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND TELECOMMUNICATION

COEP TECHNOLOGICAL UNIVERSITY

APRIL 2024

CERTIFICATE

Certified that this project report **"TerrainRover AI: Smart Remote-Controlled Car with Terrain-Aware Navigation"** is the bonafide work of "**Kedar Shiralkar, Atharva Thakur, Ujjwal Mahajan, and Pankaj Waingade**" who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Mrs. S.P. Metkar
MENTOR

Assistant Professor

Department of ENTC

COEP Technological University,

Wellesley Road, Shivajinagar, Pune- 411005

SIGNATURE

Dr. Vibha Vyas

HEAD OF THE DEPARTMENT

Department of ENTC

COEP Technological University,

Wellesley Road, Shivajinagar, Pune- 411005

SIGNATURE
Dr. Mrs. S.P. Metkar

Table of Contents:

Abstract

The TerrainRover AI project focuses on developing an intelligent remote-controlled (RC) car that can autonomously navigate and adapt to different terrains in real-time. Traditional RC cars are typically limited in their versatility due to their inability to adjust speed to varying terrain conditions, which restricts their overall performance and functionality. This project overcomes this limitation by incorporating machine learning algorithms for terrain classification, enabling the car to recognize different types of terrain and adjust its behavior, such as speed, to match the environment. By doing so, the TerrainRover AI project aims to create a more versatile and adaptable RC car that can handle a range of terrains with greater efficiency and safety.

Existing solutions for terrain recognition in RC cars include sensor-based approaches such as ultrasonic and infrared sensors, LIDAR, and camera-based systems. Ultrasonic and infrared sensors can detect obstacles but are limited in recognizing specific terrain types. LIDAR provides detailed mapping data but is expensive and requires significant computational power to process. Camera-based systems with image recognition algorithms can identify terrain types, but they demand substantial processing power and struggle in low-light or harsh weather conditions. Traditional machine learning algorithms use predefined rules and thresholds to classify terrain based on sensor data but often lack the ability to generalize to new environments. Pre-trained neural network models offer higher accuracy but may not perform well in unfamiliar environments and require significant data for training. The drawbacks of these existing solutions include limited adaptability, high computational requirements, cost and complexity, environmental sensitivity, and limited scope.

The proposed solution, Terrain Rover AI, introduces an innovative approach by utilizing a remote-controlled (RC) car with a Raspberry Pi running a machine learning algorithm to classify terrain into categories like pavement, grass, road, and sand which are detected using a Raspberry Pi Camera. The car adapts its speed based on the recognized terrain type, ensuring optimal performance and safety. The use of Raspberry Pi as the main processing unit keeps costs low while providing sufficient computational power. The system's adaptability allows it to generalize well to different and unseen environments. Enhanced safety is achieved through speed adjustments, and tailored speed improves performance and efficiency. Additionally, the system's versatility allows for various applications and future expansion possibilities, while continuous data collection can enable ongoing learning and model improvement.

Problem Statement

Design and develop a low-cost remote-controlled (RC) car that can autonomously recognize different terrain types and adjust its speed accordingly.

Motivation

The motivation behind this project lies in enhancing the capabilities of RC cars, making them versatile for exploration, surveillance, and various applications like **space rovers** and **autonomous vehicles.** Intelligent terrain-aware navigation enables the RC car to autonomously navigate through different surfaces, providing a solution for scenarios where manual control is challenging.

Objectives

The objective of the "TerrainRover: Remote Controlled (RC) Car with Terrain-Aware Navigation" project is to design, construct, and implement an autonomous RC car capable of navigating diverse terrains through the integration of machine learning and real-time control systems. This project aims to achieve the following:

- Design of 3D Prototype of RC Car
- Implementation of Terrain Classification System
- Speed Control of RC Car using Terrain Detection

Proposed Solution

The proposed solution is:

- To develop an RC Car with manual control established via Arduino and a Bluetooth module.
- To add a Raspberry Pi and a PiCamera to detect terrain based on a Machine Learning Model and send terrain types to Arduino via serial communication.
- Taking Terrain types as input, the Arduino will implement speed control by using PWM in the motor pulse.

Key Components

- RC Car Construction:
  Chassis, motors, wheels, and structural components assembly to form the RC car.
- Integration of Picamera for collecting data related to terrain conditions.
- Raspberry Pi for processing, and data exchange
- Arduino for motor control.
- Implementation of real-time control algorithms.
- Development of Python scripts for image processing using machine learning.
- Establishment of wireless and wired communication for data exchange.
- Mounting a camera on the RC car for continuous image capture.
- Convolutional Neural Network (CNN) for terrain classification:
- Training the CNN model on a dataset of various terrain images.
- Bluetooth module for manual control of RC car.
- Establishing a serial communication link between Raspberry Pi and Arduino using UART protocol.

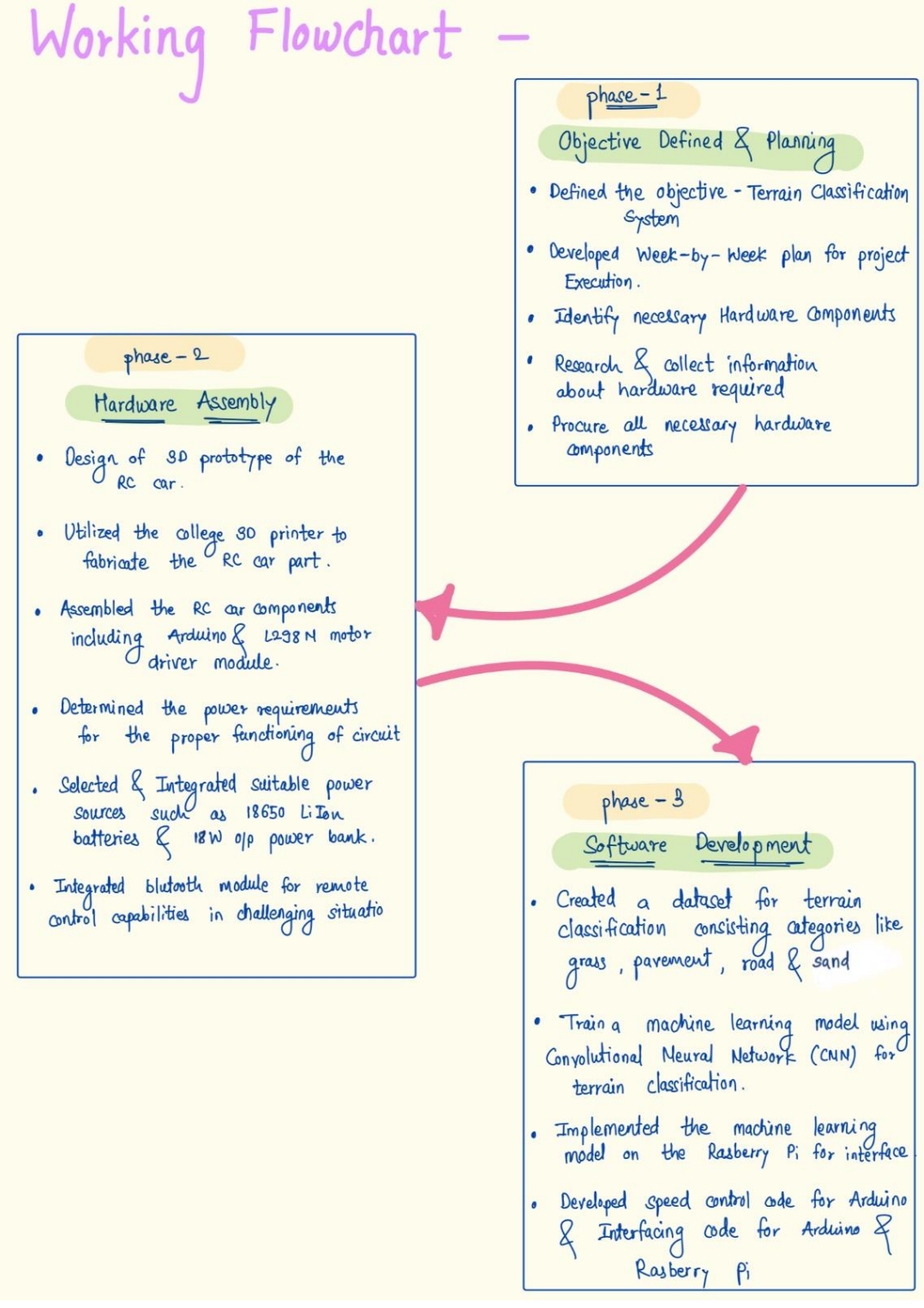
Methodology

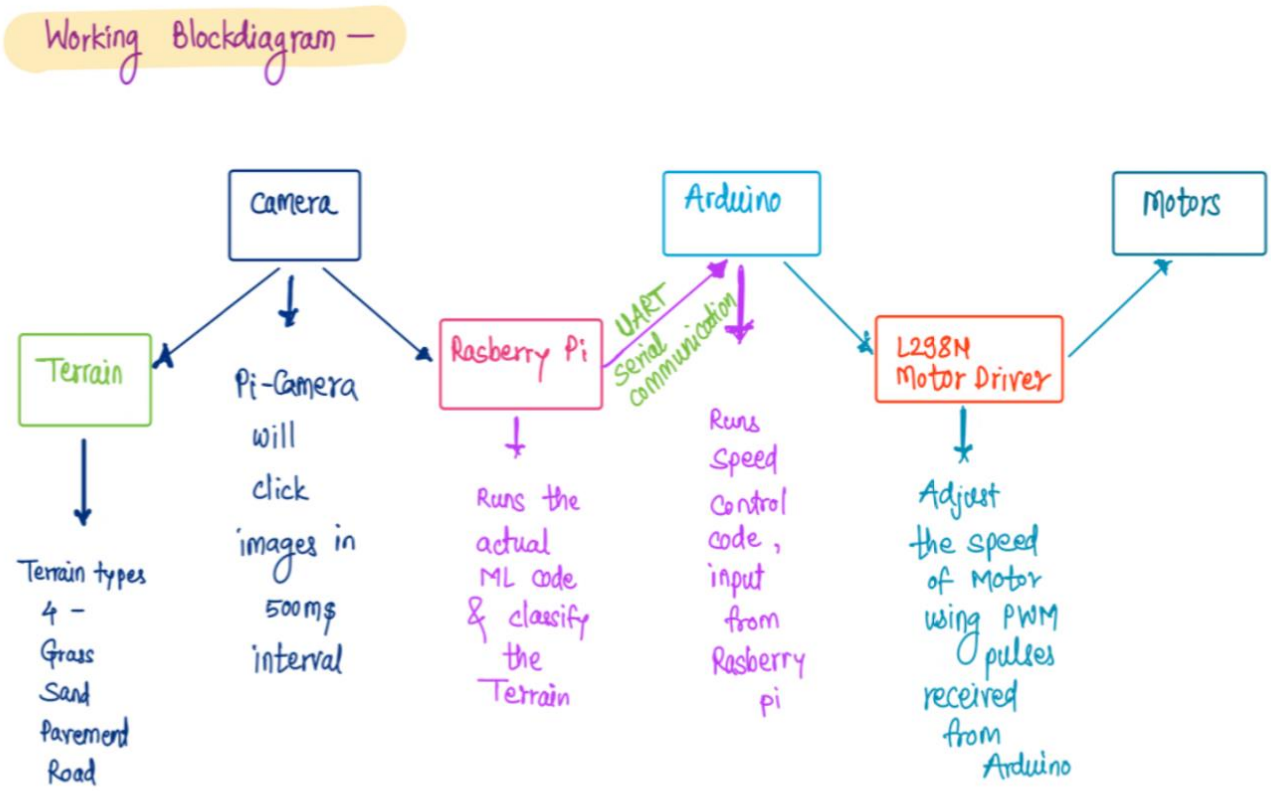
- Hardware Setup:
  - Assemble the RC car with attention to weight distribution and stability.
  - Mount camera ensuring accurate data collection.

- Software Development:
  - Implement real-time control algorithms, machine learning models, and Bluetooth communication on Arduino.
  - Dataset Collection and Model Training:
    Collect terrain images for training the CNN model.
    Train the CNN model using Python scripts on Raspberry Pi.

- Integration:
  - Interfacing Raspberry Pi and Arduino.
  - Integrate hardware and software components for seamless operation.

- Testing and Validation:
  - Conduct comprehensive testing on varied terrains to validate the system's performance and reliability.





Significance

The TerrainRover AI project holds significance in:

- Demonstrating the integration of complex hardware (RC car construction) with advanced software (machine learning and real-time control).
- Showcasing the adaptability of the RC car to diverse terrains through intelligent navigation.

Design and Simulation

Under the design phase of the project, we laid out a comprehensive approach to construct an RC car capable of recognizing different terrains using machine learning and adapting its speed accordingly. This involved selecting the appropriate hardware components and structuring them in a way that would optimize the car's performance and capabilities.

Hardware Selection and Design

i. Microcontroller Selection -

The hardware design of the project centers around the integration of two microcontrollers: Raspberry Pi and Arduino. The Raspberry Pi was chosen for its ability to handle intensive machine learning computations necessary for recognizing terrain types, while the Arduino was selected for its quick response times in motor control. By leveraging the strengths of both microcontrollers and interfacing

them using serial communication via USB cable and UART protocol, we ensured the smooth execution of commands and data flow.

ii. Motor Controller -

For motor control, we selected the L298N motor driver module due to its robustness and versatility in operating voltage (up to 45V) and its ability to drive four motors simultaneously. This module efficiently regulates the speed and direction of the motors according to terrain type. We used 150 RPM motors, providing sufficient power to drive the car across different surfaces.

iii. Power Management System -

For power management, we employed an 18W power bank with a maximum load capacity of 10000 mAh to supply a stable and constant power source to the Raspberry Pi. This choice leveraged the power bank's built-in protection circuits to safeguard the system. The L298N and Arduino were powered using two 18650 Li-Ion rechargeable batteries with a combined output voltage of 7.4V, meeting the L298N's requirements.

iv. Camera -

In terms of component selection, we used a 5MP, 1080p Raspberry Pi camera according to our project requirements and its low cost. This camera provided the necessary quality and functionality for terrain recognition, and it could be easily mounted and adjusted for angle using a simple wooden stick structure and a camera mount on the upper 3D-printed layer.

v. Communication between Arduino and Raspberry Pi -

The baud rate for serial communication was set at 9600, allowing for reliable data transfer between the Raspberry Pi and Arduino. This can be adjusted to a higher rate such as 115200 to further minimize latency if necessary.

vi. Chassis Construction -

In the design phase of our project, we approached the construction of the RC car from scratch with the intent to explore a variety of technologies and integrate our learning from courses into a practical application. To achieve this, we designed a custom two-layer chassis, optimizing for both lightweight structure and functional requirements. The upper layer of the chassis was dedicated to placing the camera and Raspberry Pi, while the lower layer was reserved for the Arduino, batteries, motors, and motor control system.

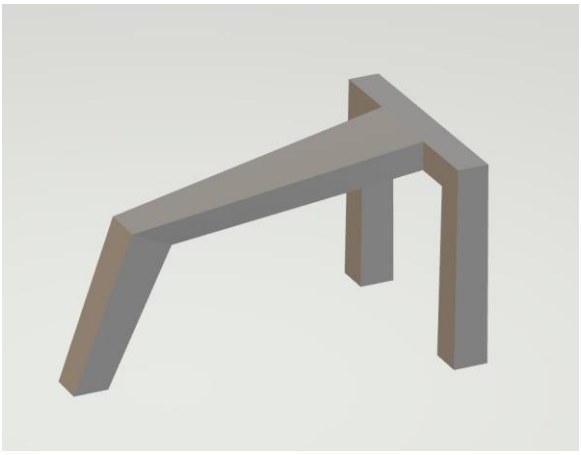
We constructed the lower layer of the chassis from lightweight softwood, which provided a sturdy base for the motors and motor control system. We used L-channel plates to mount the motors on the base plate, ensuring they remained fixed and maintained the correct direction. The base plate was cut to the desired dimensions using a hand saw.

The upper layer was designed for the camera and Raspberry Pi placement. We used 3D printing technology to create a structure for camera placement using an FDM 3D printing machine with ABS material in the Production and Manufacturing department of our college. The camera was then mounted on this 3D-printed structure, which was attached to an acrylic sheet for stability. The acrylic sheet was cut using a laser cutter at the production department. We fitted the 3D-printed part on the acrylic sheet

using glue and screws and then attached the upper layer to the lower wooden base layer using long-sized screws and bolts.

The assembly of the car included securing the Arduino and motor driver onto the chassis using small screws, guided by a rough placement diagram that we designed at the outset of the project. We drilled the base layer according to the measurements and attached the motor driver and Arduino in the designated positions. The batteries were mounted on the base layer using double-sided tape, ensuring they remained secure during operation. By selecting 65mm-sized wheels, we achieved proper ground clearance for our requirements.

The hardware design of the project involved a meticulous selection of components and careful integration to achieve a seamless and efficient system that could handle both the computational and operational demands of the project. Through this process, we not only implemented innovative design approaches but also gained hands-on experience with new technologies like 3D printing and laser cutting, enriching our learning experience and contributing to the success of the project



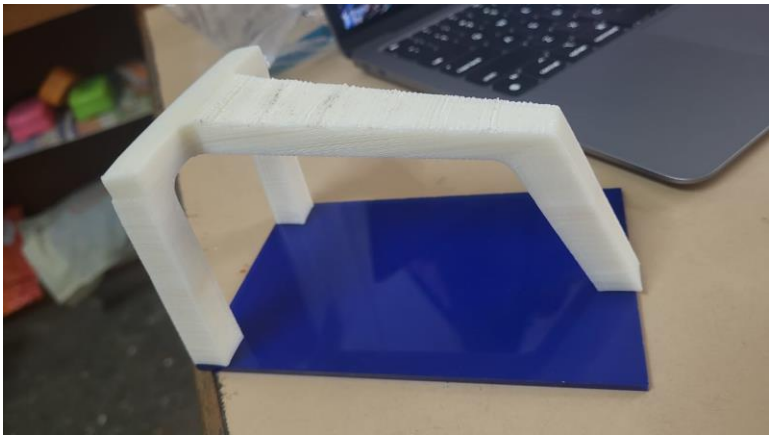


Fig: Basic Structure of 3D printed part

vii. Software and Data Handling -

The design also took into account software implementation, specifically machine learning algorithms for terrain recognition. By processing camera data, the Raspberry Pi uses these algorithms to classify the terrain type and send the appropriate commands to the Arduino. This relationship ensures that the RC car can adjust its speed based on the current terrain type, providing optimal performance and safety.

Components Used

No.	Component	Specifications	Quantity
1	Raspberry Pi Model 4B	• Processor: Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz. • Memory: • Available options of 2GB, 4GB, or 8GB LPDDR4 SDRAM. • Connectivity: • Wireless: Dual-band 802.11ac Wi-Fi, Bluetooth 5.0, BLE. • Ethernet: Gigabit Ethernet. • USB: Two USB 3.0 ports and two USB 2.0 ports. • GPIO: 40-pin GPIO header for interfacing with sensors and peripherals. • Video Output: • Dual micro-HDMI ports supporting up to 4Kp60 resolution. • Ports: • USB-C power input. • 2 × micro-HDMI ports. • 3.5mm audio jack. • Storage: Supports microSD card for storage and booting. • Power: Requires a 5V, 3A power supply via USB-C port.	1
2	Arduino Uno	• Microcontroller: ATMega328P. • Memory • AVR CPU at up to 16 MHz • 32KB Flash • 2KB SRAM • 1KB EEPROM • Security • Power On Reset (POR) • Brown Out Detection (BOD) • Peripherals • 2x 8-bit Timer/Counter with a dedicated period register and compare channels • 1x 16-bit Timer/Counter with a dedicated period register, input capture, and compare channels • 1x USART with fractional baud rate generator and start-of-frame detection • 1x controller/peripheral Serial Peripheral Interface (SPI) • 1x Dual mode controller/peripheral I2C • 1x Analog Comparator (AC) with a scalable reference input • Watchdog Timer with separate on-chip oscillator • Six PWM channels Interrupt and wake-up on pin change	1

Components Used

No.	Component	Specifications	Quantity
3	L298N motor driver module	• Voltage Supply: • Logic voltage: 5V. • Motor voltage: 5V to 45V (depending on the motors being driven). • Maximum Output Current: • 2A per channel (continuous). • Up to 3A per channel (peak, for a short duration). • Channels: • Dual H-bridge design, capable of controlling two DC motors or one stepper motor. • PWM Control: • Supports pulse-width modulation (PWM) for speed control of DC motors. • Thermal and Overcurrent Protection: • The module features thermal shutdown and overcurrent protection to safeguard the motors and the driver itself. • Logic Level Inputs: • Compatible with TTL logic levels (0V - LOW, 5V - HIGH). • Four input pins (IN1 to IN4) for controlling the direction of the two channels. • Two enable pins (ENA, ENB) for controlling the speed of the motors via PWM. • Output Pins: • Four output pins (OUT1 to OUT4) for connecting the motors. • Heat Sink: • The module often comes with a built-in heat sink for improved thermal performance during high-current operations. • Mounting and Connectivity: • Typically comes with screw terminals for easy connection to motors and power supply. • Standard pin headers for easy integration with microcontrollers and other electronic components.	1
4	BO motors of 150 RPM	Voltage: Operating voltage range: Typically 3V to 12V. Torque and Speed: Speed: Approximately 150 RPM (revolutions per minute) at 12V. Torque: Varies depending on the model and manufacturer, but typically around 1.2 kg/cm to 1.5 kg/cm at 12V.	4

Components Used

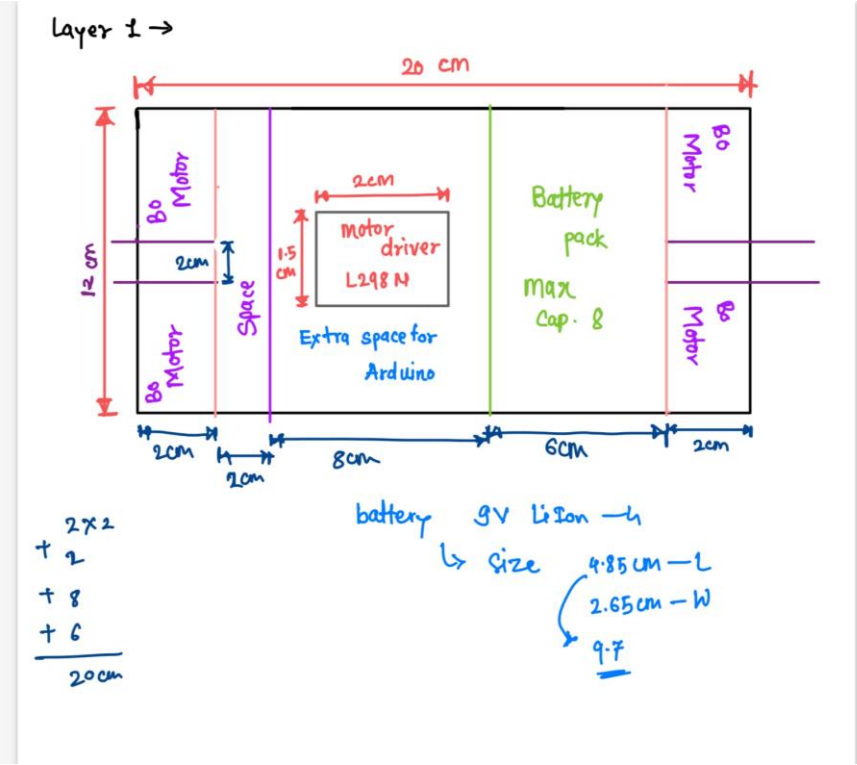
No.	Component	Specifications	Quantity
5	PiCamera	• **Graphics Coprocessor:** AMD FirePro 2270 • **Brand:** Raspberry Pi • **Graphics RAM Size:** 0.09 • **Graphics RAM Type:** GDDR3 • **Included Components:** PI 5MP Camera Board Module • 5 megapixel native resolution sensor-capable of 2592 x 1944 pixel static images • Supports 1080p30, 720p60 and 640x480p60/90 video • Camera is supported in the latest version of Raspbian, Raspberry Pi's preferred operating system • For Raspberry Pi zero/zero W/Zero WH kindly use a different camera cable	1
6	HC-05 Bluetooth Module	• **Bluetooth Version:** Supports Bluetooth 2.0 + EDR (Enhanced Data Rate) for wireless communication. • **Operating Voltage:** Compatible with a voltage range of 3.3V to 5V, making it suitable for most microcontroller projects. • **Communication Range:** Offers a range of up to 10 meters for wireless data transmission. • **Communication Interface:** Utilizes a UART (Universal Asynchronous Receiver-Transmitter) interface for serial communication with microcontrollers.	1

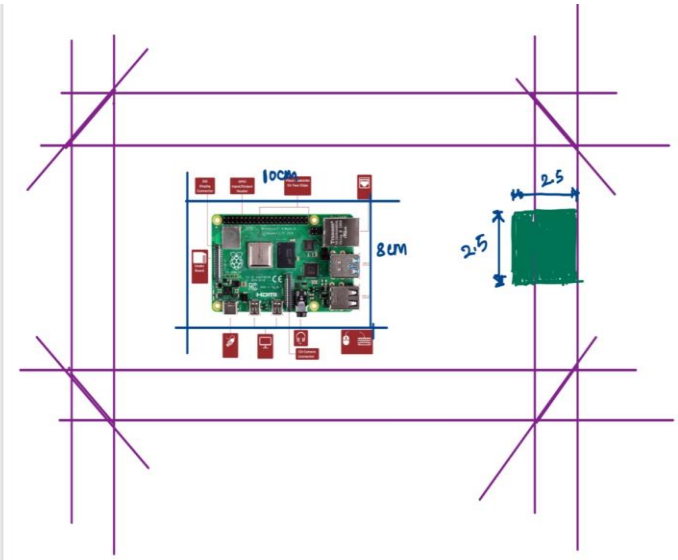
7	MI Power Bank(18W)	• **Model:** PLM13ZM
		• **Input:**
		  • **(USB-C):** 5V-2.1A , 9V-2.1A,12V-1.5A
		  • **(Micro-USB)** : 5V-2.1A , 9V-2.1A , 12V-1.5A
		• **Output** :
		  • **(Dual-port USB-A)**: 5.1V - 2.6A
		  • **(Single-Port USB-A)**:5.1V-2.4A,9V-2A, 12V-1.5A
		• **Cell Capacity** : 37Wh,3.7V, **10000mAh**
		• **Made in India**
			1

8	18650 Li-Ion Rechargeable batteries	• **Capacity:** 2200 mAh • **Standard Voltage:** 3.7 V	2

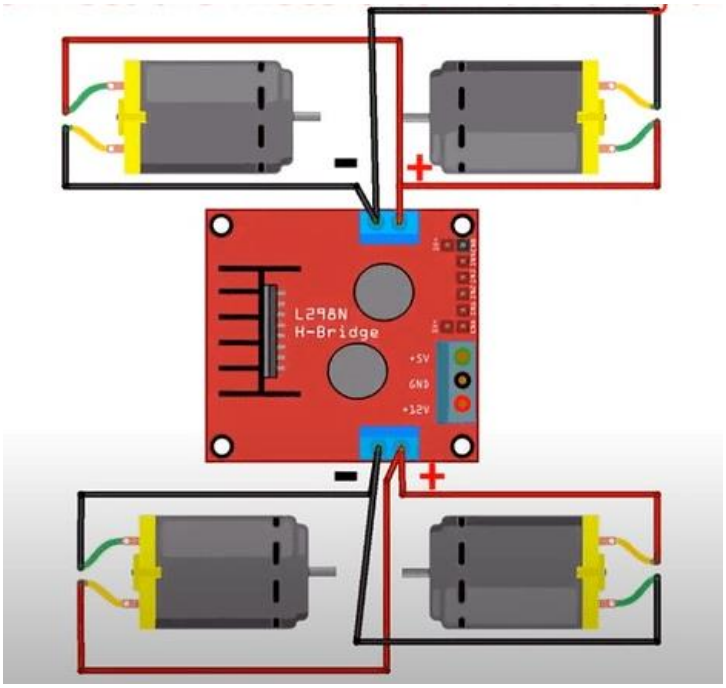
Components Used			
No.	**Component**	**Specifications**	**Quantity**
9	Battery Holder	• **Material:** High Quality Plastic • **Compatibility:** 18650 Batteries • **Output Voltage:** 7.4V	1
10	Battery Charger	**Model:** SP Electron Single Slot Lithium Battery Charger **Compatibility:** Suitable for 10440, 14500, 16340, 16650, 18350, 18500, 18650, 26650 Batteries **Output Voltage:** 4.2V **Current Rating:** 0.5 A	

Here are some designs for the arrangement of these components –

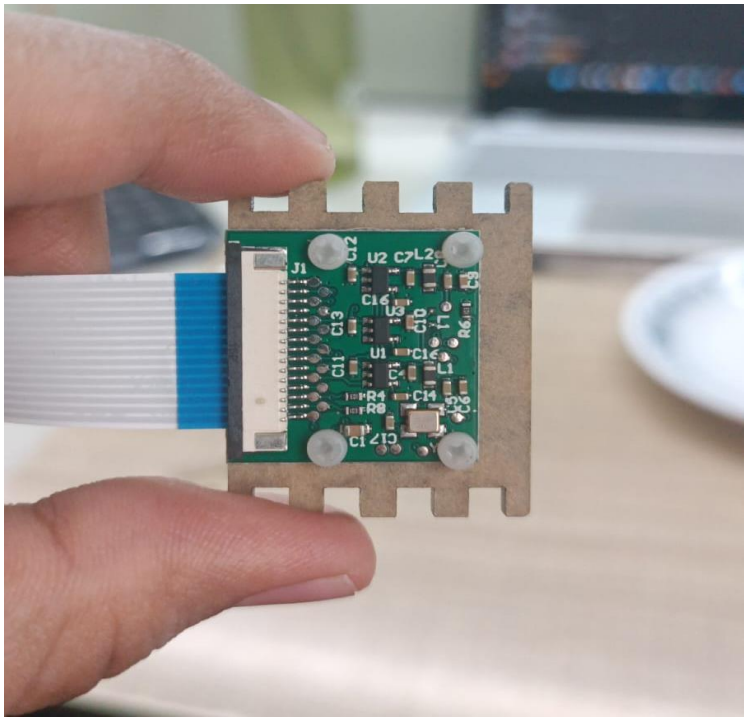




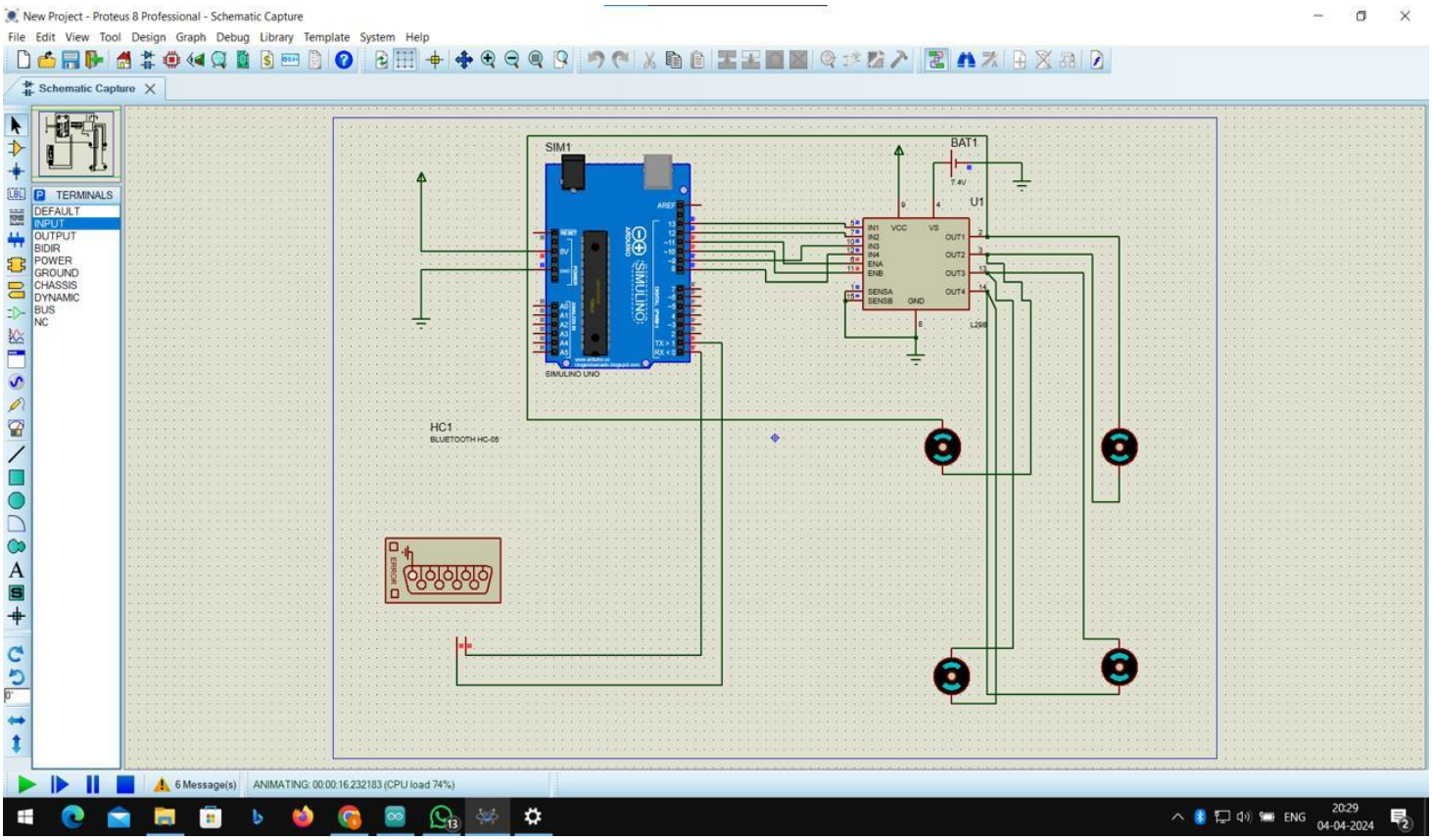
Reference Circuit diagram for motor connection –



Camera module mount -



Simulation



Assembly

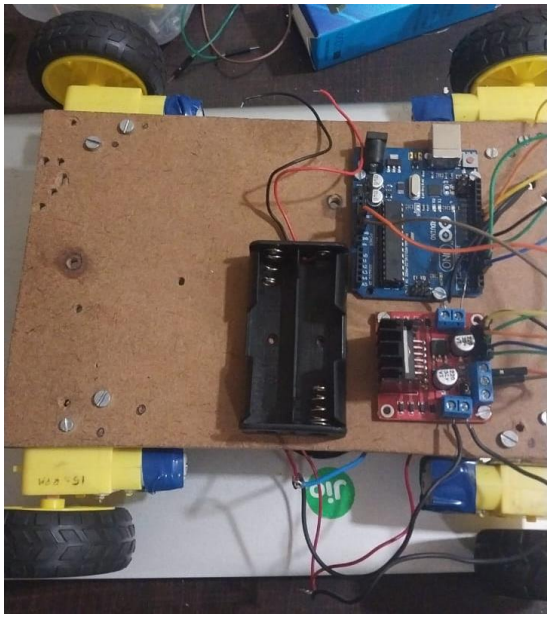


Fig: Basic Assembly

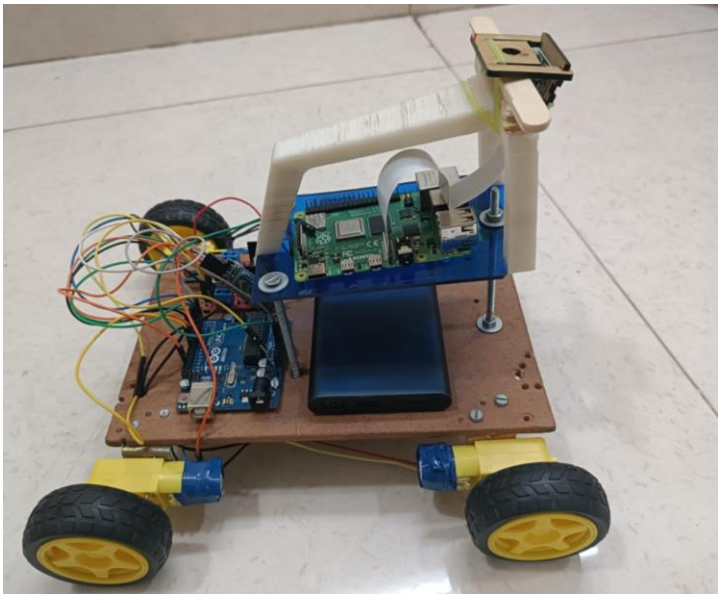
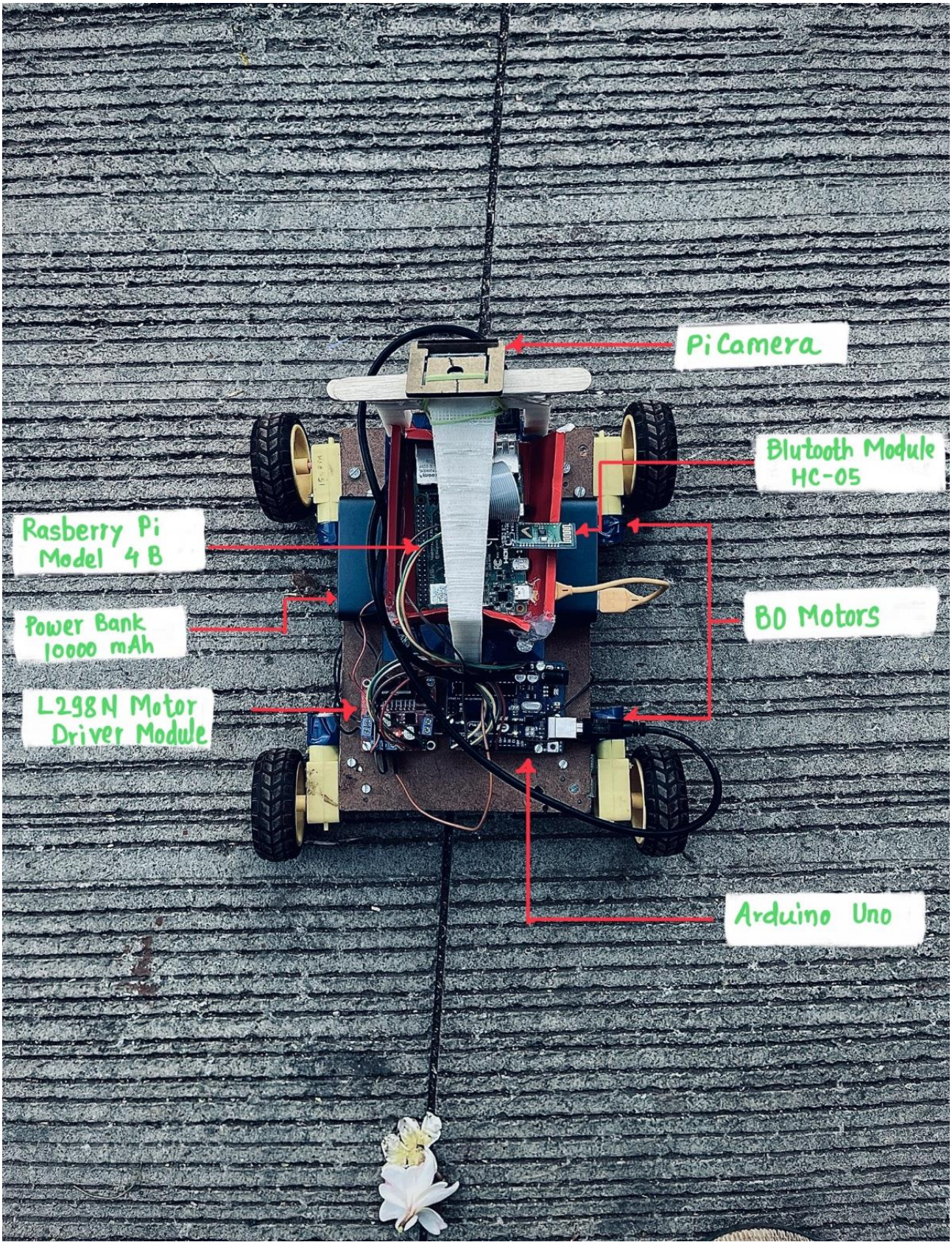


Fig: Full assembly

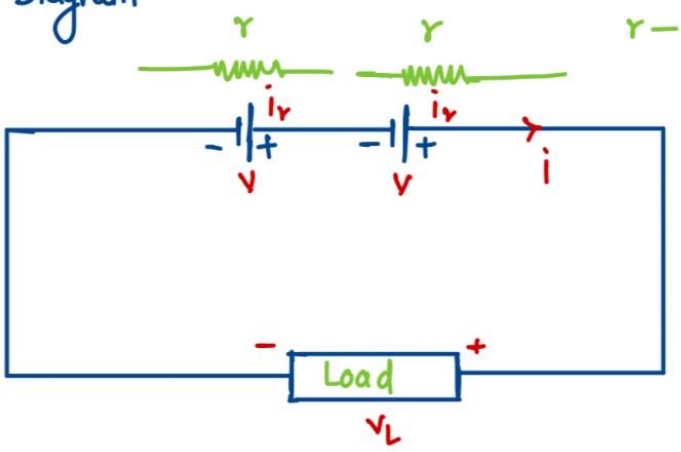
Final Prototype



Calculations

Effective Current Calculations for Power System –

Circuit Diagram –



by using KVL –

$$-V_L + V + V = 0$$

$$V_L = 2V$$

inside battery,

$$V = (i_r) \cdot r$$

$$V_L = 2(i_r) \cdot r$$

$$V_L = i_r (2r)$$

i_r – current same as one battery

2r – effective due resistance to 2 batteries

Software Part (Arduino)

Arduino Code for Automatic Speed Control –

```
    digitalWrite(in4, LOW);

    analogWrite(en1, LOW); // PWM Speed Control (0 to 255)
    analogWrite(en2, LOW);
}

void loop() {
    if (Serial.available() > 0) {
        rpi = Serial.readStringUntil('\n');

        if (rpi == "G") {
            // Move forward with full speed on grass
            analogWrite(en1, 175);
            analogWrite(en2, 175);
            digitalWrite(in1, HIGH);
            digitalWrite(in2, LOW);
            digitalWrite(in3, HIGH);
            digitalWrite(in4, LOW);
            Serial.println("Grass");
            delay(4000);
            digitalWrite(in1, LOW);
            digitalWrite(in2, LOW);
            digitalWrite(in3, LOW);
            digitalWrite(in4, LOW);

        } else if (rpi == "P") {
            // Move forward with reduced speed on pavement
            analogWrite(en1, 220);
            analogWrite(en2, 220);
            digitalWrite(in1, HIGH);
            digitalWrite(in2, LOW);
            digitalWrite(in3, HIGH);
            digitalWrite(in4, LOW);
            Serial.println("Pavement");
        } else if (rpi == "S") {
            // Move forward with a different speed on sand
            analogWrite(en1, 200);
            analogWrite(en2, 200);
```

Explanation -

- Initialization:
    - In `setup()`, configure serial communication at 9600 baud.
    - Set motor control pins (`en1`, `en2`, `in1`, `in2`, `in3`, `in4`) as outputs.
    - Start with all control pins set to low (car stationary).

- Main Loop:
    - Constantly listen for serial input for a new command.
    - Read input until newline (`\n`).
    - Interpret the first character as terrain type.

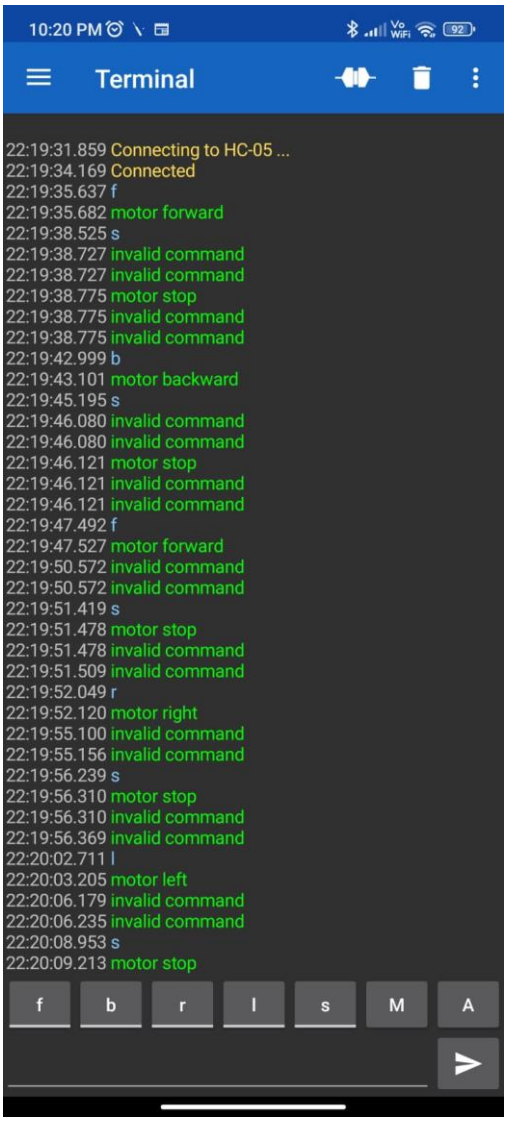
- Command Handling:
    - "G" (Grass): Move forward at full speed by setting PWM pins (`en1`, `en2`) to 255 and control pins (`in1`, `in3`) to high, (`in2`, `in4`) to low.
    - Other Terrains: Adjust speed for different terrains (e.g., pavement, sand, road).
    - Invalid Commands: Stop the car by setting all motor control and PWM pins to low.
    - Speed Control with PWM:
    - The car's speed is controlled using Pulse Width Modulation (PWM) by adjusting PWM pulse values given to enable pins of motor control.
    - The relationship between PWM value and motor RPM is as follows:
        - 255 PWM Value: Motors run at 150 RPM.
        - 220 PWM Value: Motors run at 129.41 RPM.
        - 200 PWM Value: Motors run at 117.64 RPM.
        - 175 PWM Value: Motors run at 102.94 RPM.

- Safety:
    - Ensure immediate stop in case of invalid or unrecognized commands.

Arduino Code for Manual Control –

```
  analogWrite(en2, 255);
}

void loop() {
  if (bluetooth.available() > 0) {
    char command = bluetooth.read();
    switch (command) {
      case 'r':
        Serial.println("motor right");
        // Turn the motor on in one direction
        digitalWrite(in1, HIGH);
        digitalWrite(in2, LOW);
        digitalWrite(in3, HIGH);
        digitalWrite(in4, LOW);
        delay(3000); // Run the motor for 3 seconds
        break;
      case 'l':
        Serial.println("motor left");
        // Turn the motor on in one direction
        digitalWrite(in1, LOW);
        digitalWrite(in2, HIGH);
        digitalWrite(in3, LOW);
        digitalWrite(in4, HIGH);
        delay(3000); // Run the motor for 3 seconds
        break;
      case 'b':
        Serial.println("motor backward");
        // Turn the motor on in one direction
        digitalWrite(in1, LOW);
        digitalWrite(in2, HIGH);
        digitalWrite(in3, HIGH);
        digitalWrite(in4, LOW);
        delay(3000); // Run the motor for 3 seconds
        break;
      case 'f':
        Serial.println("motor forward");
        // Turn the motor on in one direction
```



Explanation -

- Objective: Control an RC car using Bluetooth communication.

- Initialization:
- Utilizes `SoftwareSerial library to create a virtual serial port on Arduino's RX (pin 0) and TX (pin 1) pins.
    - Initializes serial and Bluetooth communication at 9600 baud.
    - Configures motor control pins (`en1`, `en2`, `in1`, `in2`, `in3`, `in4`) as outputs.
    - Stops motors initially and sets the motor speed to maximum (255) using PWM.
- Main Loop:
    - Continuously checks for incoming Bluetooth commands.

    - Reads available commands and uses a `switch` statement to execute actions based on the command received.

- Command Handling:
    - 'r' (Right): Sets the control pins to turn the car right for 3 seconds.
    - 'l' (Left): Reverses motor direction to turn the car left for 3 seconds.
    - 'b' (Backward): Moves the car backward.
    - 'f' (Forward): Moves the car forward.
    - 's' (Stop): Halts the car's movement.

- Post-Execution:
    - Pauses motor operation for 3 seconds after executing a command.
    - Invalid commands result in a message "invalid command" and no action is taken.

- Outcome:

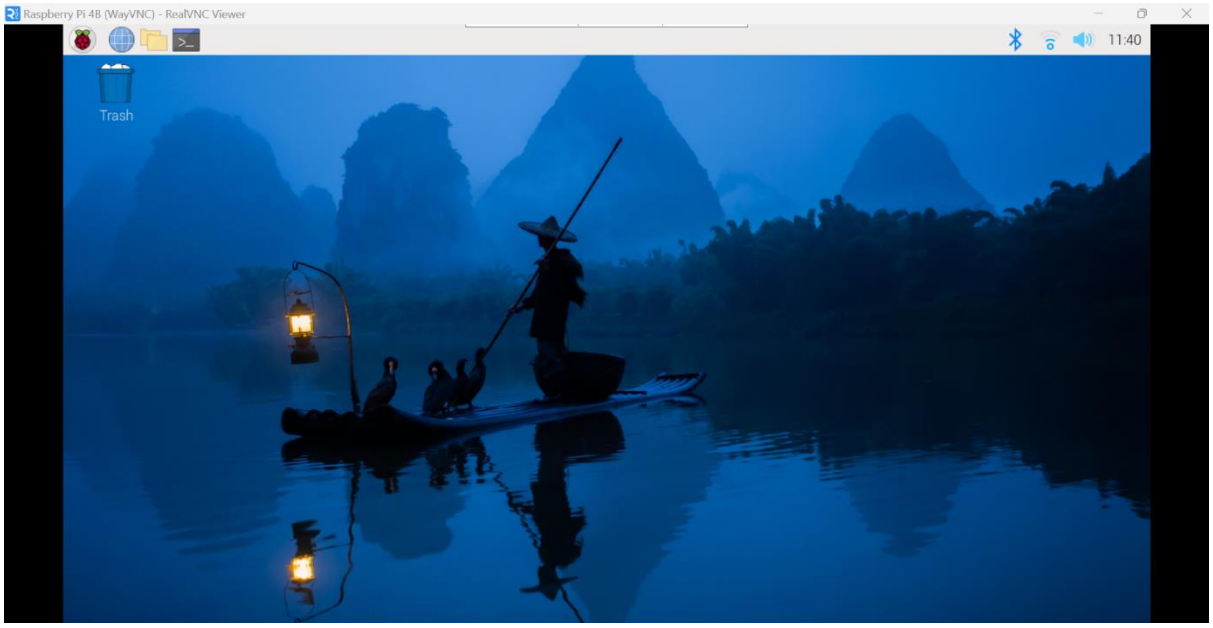
Allows the user to remotely control the RC car through Bluetooth communication, enabling various maneuvers such as moving forward, backward, left, right, and stopping as needed.

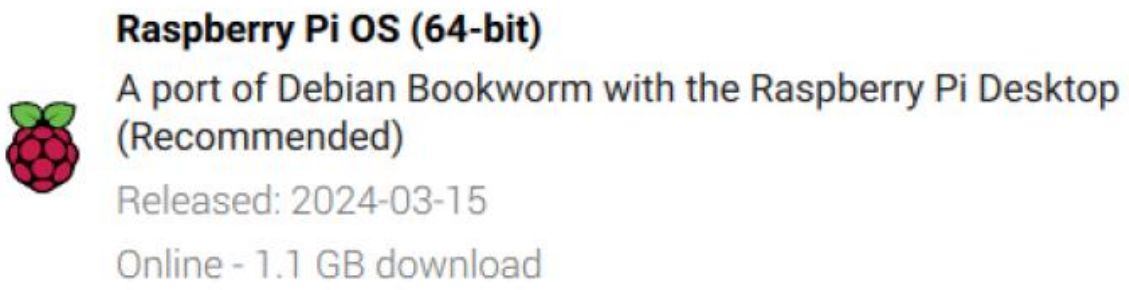
Software Implementation on Raspberry Pi :

1. Raspberry Pi Setup

- **GUI and OS Installation:**

- We used the Raspberry Pi 4B model with 8Gb RAM for image processing in our application.
- For fast and efficient working, we installed the Bookworm 64-bit OS using the Raspberry Pi Imager on a 32Gb SD Card.
- We provided a Wi-Fi SSID and password during installation so that as soon as the Pi boots up, it can be connected to the Wi-Fi, and VNC (Virtual Network Computing) software on a laptop can be used to directly display the Raspberry Pi graphics as long as both devices are on the same internet network. This ensures that there is no need for a wired connection for the monitor, mouse, and keyboard. For this, first, the IP address of the Raspberry Pi was found when it was connected to my home router, and then it was used to access the Pi via SSH to enable VNC and then set it up.
- I also later set up my mobile hotspot in such a way that I could connect to it whenever my home Wi-Fi was not available. This enabled us to operate VNC anywhere we went.

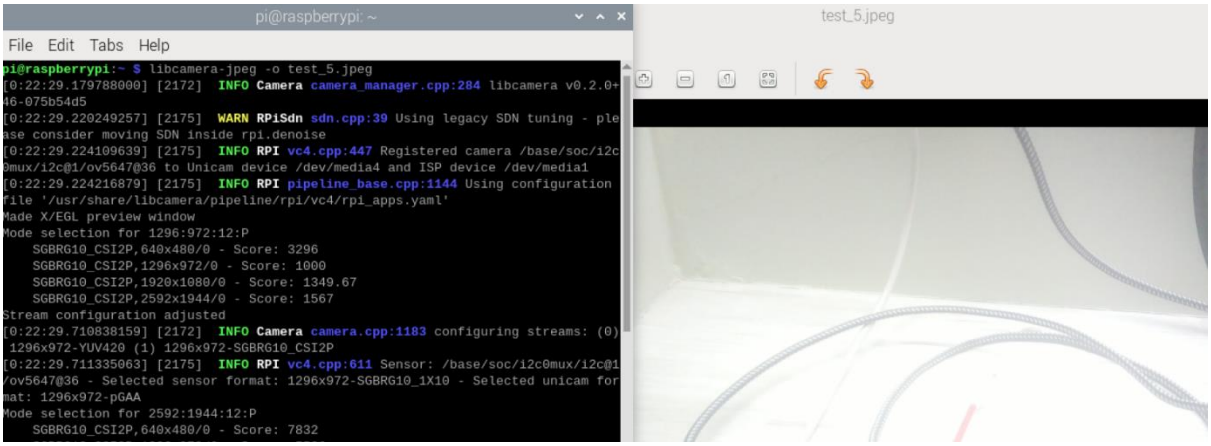




- **<u>Camera Setup:</u>**

- For image capture, we used the Raspberry Pi 5Mp Camera Module with high-definition image quality.
- With the new libcamera stack in the Raspberry Pi OS, cameras are set to auto-connect to the Pi without any enabling.
- This stack has its own set of commands in the terminal and a different library which is compatible with Python.
- The Camera was connected using the CSI port on the Raspberry Pi.
- Then, the Camera was tested using different Linux commands, and a test image was taken to verify the functionality.

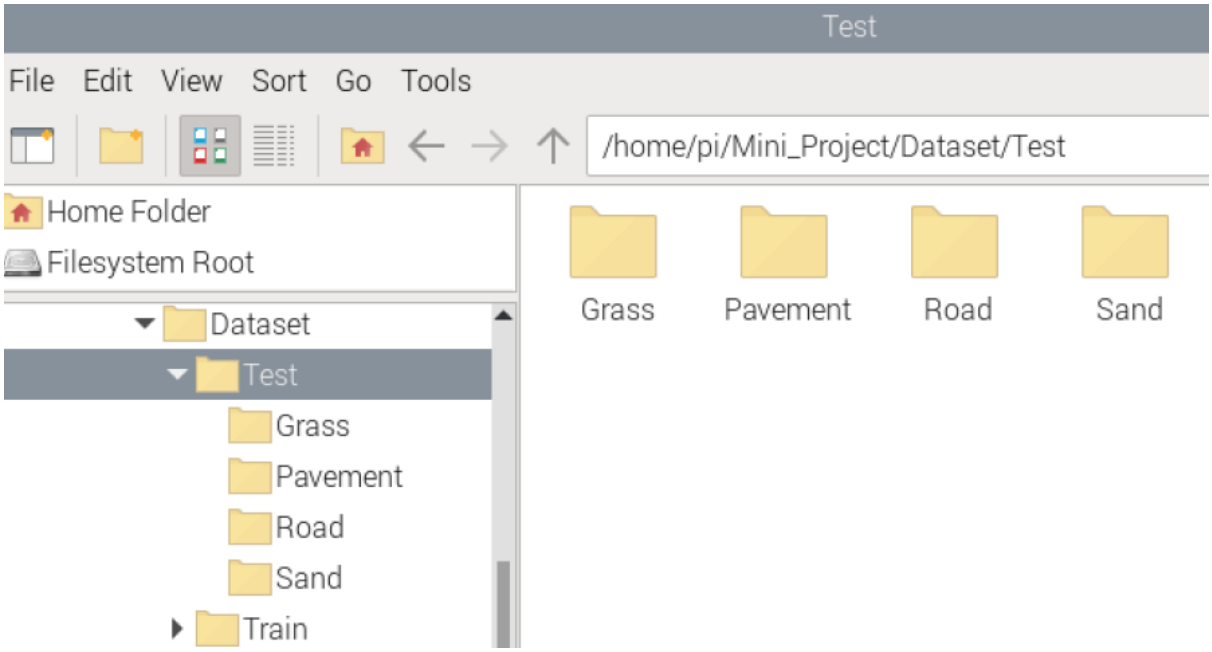
- For Python programming, the Camera needs the latest PiCamera2 library for the libcamera stack instead of the PiCamera library which is only compatible with the legacy Camera Stack.



2. Dataset Collection

- **Dataset Structure:**

- We wanted to collect a Dataset of 4 classes:
    1. Grass
    2. Pavement
    3. Road
    4. Sand
- So, we decided to collect a total of 4000 images with 1000 images per class.
- We designed it with a 600:400 Train/Test Split for the training and evaluation Dataset.
- The directories were designed in such a way that the Dataset folder will consist of 2 folders called Train and Test, and each of them will consist of images of all 4 classes in 4 different folders for each class.



- **<u>Camera Script in Python for Dataset Capture:</u>**

- We designed a script that will set up the camera, show us the camera preview, capture the image, and send it to the specified directory, even the capturing frequency of images can be set (e.g. Every 200ms), and the total duration of the captures can be specified.

- Thus, for example, to create the train directory, we would set a directory for each of the classes that will be created in the code using the os library in Python, and if it is the train set, we need to capture 600 images, so the intervals will be set at 200ms, and the duration will be 120 seconds, which will give us 600 images of the specified terrain.

- We used the PiCamera2 library, the os library, and the time library for setting delays.

```
Camera_2.py > ...
  5  def capture_images(duration, interval, output_dir):
 31
 32      # Stop the preview
 33      camera.stop_preview()  # Make sure this is called after the preview has been st
 34
 35  if __name__ == "__main__":
 36      # Specify the duration (in seconds) and interval (in seconds)
 37      duration = 80
 38      interval = 0.2
 39
 40      # Specify the output directory
 41      output_dir = "/home/pi/Test_cp/Test_1"
 42
 43      # Capture images
 44      capture_images(duration, interval, output_dir)
 45

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

[0:49:05.916984525] [3314]  INFO Camera camera.cpp:1183 configuring streams: (0) 1920x1080-BG
640x480-YUV420 (2) 1920x1080-SGBRG10_CSI2P
[0:49:05.917579892] [3337]  INFO RPI vc4.cpp:611 Sensor: /base/soc/i2c0mux/i2c@1/ov5647@36 -
sensor format: 1920x1080-SGBRG10_1X10 - Selected unicam format: 1920x1080-pGAA
QStandardPaths: wrong permissions on runtime directory /run/user/1000, 0770 instead of 0700
Captured image: /home/pi/Test_cp/Test_1/image_0.jpg
Captured image: /home/pi/Test_cp/Test_1/image_1.jpg
Captured image: /home/pi/Test_cp/Test_1/image_2.jpg
Captured image: /home/pi/Test_cp/Test_1/image_3.jpg
Captured image: /home/pi/Test_cp/Test_1/image_4.jpg
Captured image: /home/pi/Test_cp/Test_1/image_5.jpg
Captured image: /home/pi/Test_cp/Test_1/image_6.jpg
                                                    Select Indentation
```

3. Model Selection and Training

- **Algorithm Selection:**
    - For the image classification task, we used a Convolutional Neural Network (CNN) architecture.
    - The reason for this selection was that CNN has an innate ability to extract features from an image. For other algorithms, you might have to use some computer vision global descriptors to extract those features to get a decent accuracy of classifications.
    - Here is some information about CNNs:
    - **What are CNNs?**
        - **CNNs** are a class of deep neural networks specifically designed for processing grid-like data, such as images.
        - They are inspired by the visual processing in the human brain, where neurons respond to specific regions of the visual field.
        - CNNs excel at capturing local patterns and hierarchical features within images.
    - **Why are CNNs Ideal for Image Classification?**
        - **Automatic Feature Learning**: Unlike traditional methods that require hand-crafted features, CNNs automatically learn features from raw data. This ability to learn complex and abstract features directly from the data is crucial for image classification.
        - **Spatial Hierarchies**: CNNs can automatically learn spatial hierarchies of features, including edges, textures, and shapes. These features are essential for recognizing objects in images.
        - **Hierarchical Layers**: CNNs consist of interconnected layers that process input data step by step. Key layers include:
            - **Convolutional Layers**: These layers apply filters to detect specific patterns (such as edges) in the input data. Feature maps are generated by sliding these filters over the data.
            - **Pooling Layers**: These reduce feature map size, making the model computationally efficient. Pooling helps retain important information while discarding less relevant details.
            - **Fully-Connected Layers**: These layers connect all neurons to learn non-linear combinations of features.
            - **Softmax Layer**: The final layer produces a probability distribution across possible class labels.
    - **Visualizing CNNs**:
            - Imagine a CNN as a stack of interconnected layers, each responsible for specific tasks like feature extraction, abstraction, and classification.
            - The input image passes through these layers, gradually transforming it into a representation suitable for classification.

- **Why CNNs Over Other Models?**
    - **Human Supervision Not Required**: CNNs automatically detect important features without manual intervention.
    - **Effective on Raw Data**: CNNs learn directly from raw pixel values, making them robust to variations in lighting, scale, and orientation.
    - **State-of-the-Art Performance**: CNNs consistently achieve top performance in image classification benchmarks.

- **<u>Model Selection:</u>**

    - We went through various research papers and did some research about suitable models for this task and came up with around 6 models before we found our best fit.

- Here is the progression of the model architecture:

    - The initial **CNN architecture** was a more complex architecture with a (224,224) input image size and dropout layers to prevent overfitting. A CNN comprises two main components: the **feature extraction** layers and the **classifier** layers.
      The features section includes two **convolutional layers** with ReLU activation, followed by **max-pooling**. These layers learn hierarchical features from input images, detecting edges, textures, and patterns. The output is a feature map capturing relevant information. The classifier part processes the feature map, flattening it and adding a **dropout layer** to prevent overfitting. It then includes two fully connected layers (with ReLU activation) to learn class-specific representations. The final layer applies **softmax** to produce probabilities for each class. Transformations like random horizontal flips and random rotation were applied to create a more diverse dataset. Categorical Cross entropy was used along with the Adadelta optimizer. This model initially gave an accuracy of **89.6%.**
    - Later, various hyperparameters were tuned including the number of epochs and learning rates, and the accuracy went up to 96.5%. However, the latency was quite high, it came out to be around 350ms.

    - Finally, after observing the confusion matrix and the model latency, and understanding that the above model is more suitable when intricate features need to be detected, we decided that a simpler model architecture might suffice for this task.

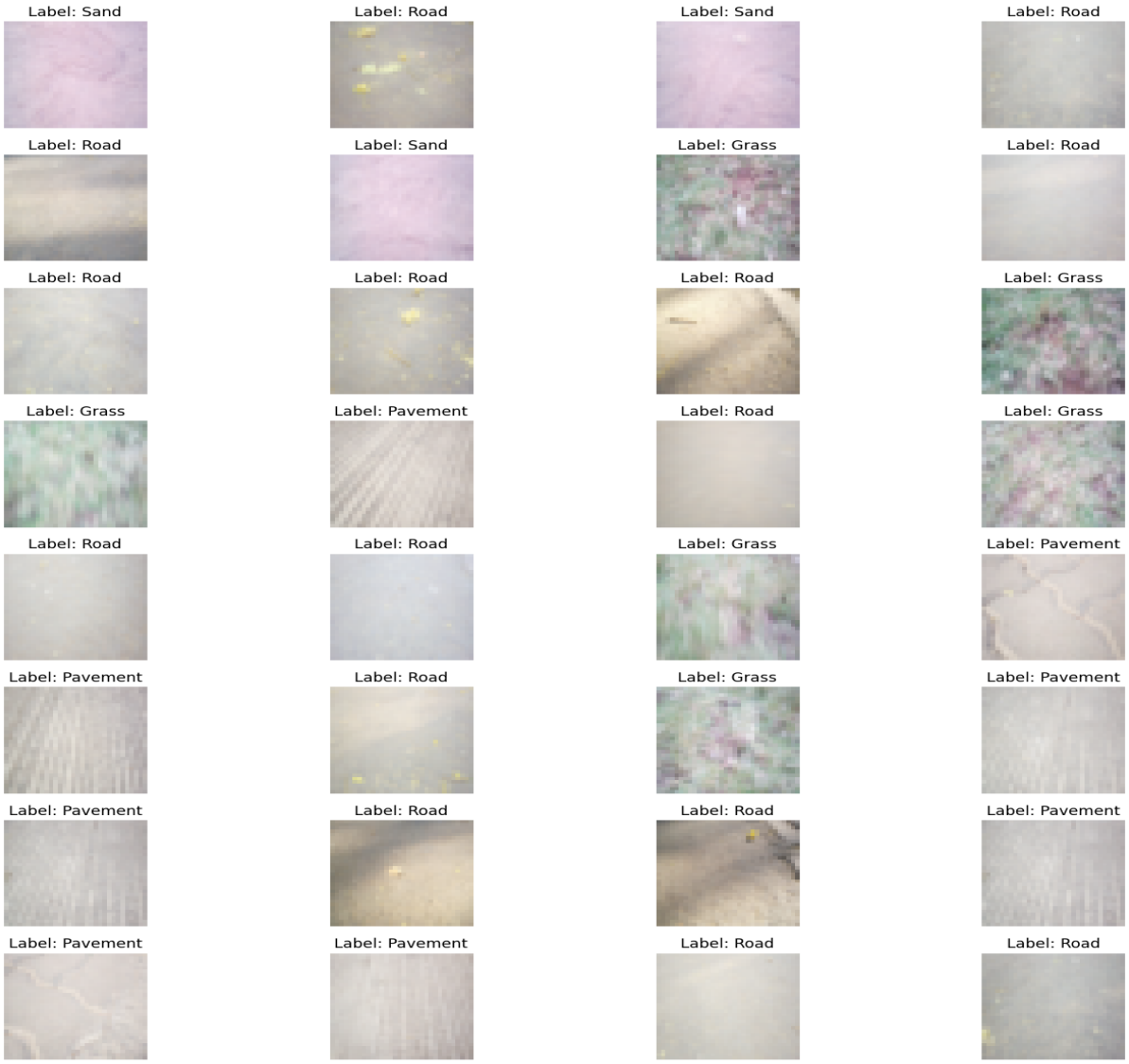
    - Thus, we removed the dropout layer, reduced the input image size to (32, 32) pixels, and simplified the architecture. We used Adam as an optimizer instead of Adadelta. This model gave an accuracy of 99.31%. The most incredible part was the latency. From the previous 350ms, it came down to 8ms on the Raspberry Pi.

Following is the progression of the model design and the resultant accuracy and latency:

Model Number	**Testing Accuracy**	**Latency on Raspberry Pi**	**Corrections made compared to the previous model**
1	89.69%	370 ms	
2	94.33%	360 ms	Increased the number of epochs from 25 to 40 to find the accuracy saturation point.
3	96.56%	350 ms	Increased learning rate from 0.001 to 0.005 and changed kernel size for pooling.
4 (Final)	99.31%	8 ms	Removed dropout layers, changed input image size from (224,224) to (32,32), reduced the number of pooling layers due to simpler architecture and lesser input parameters.

4. Final Model Training and Evaluation:

- After initial preprocessing and transformations, a code was written to label the dataset as it was being loaded onto the model, the following was its output:



- Then, the model architecture was defined, and the loss function and optimizer were defined:

```
TerrainClassifier(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=2048, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=4, bias=True)
)
```

```python
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

- After 10 epochs of training, the following results were achieved:

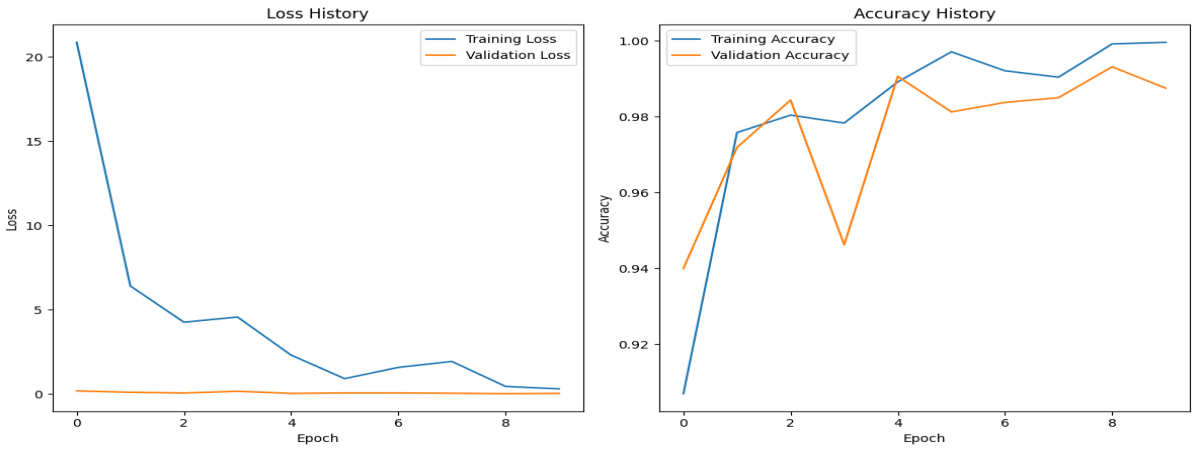
```
Epoch [1/10], Training Loss: 20.8615, Training Accuracy: 90.70%, Validation Loss: 0.1812, Validation Accuracy: 94.00%
Epoch [2/10], Training Loss: 6.4065, Training Accuracy: 97.58%, Validation Loss: 0.1005, Validation Accuracy: 97.19%
Epoch [3/10], Training Loss: 4.2582, Training Accuracy: 98.04%, Validation Loss: 0.0564, Validation Accuracy: 98.44%
Epoch [4/10], Training Loss: 4.5639, Training Accuracy: 97.83%, Validation Loss: 0.1587, Validation Accuracy: 94.62%
Epoch [5/10], Training Loss: 2.3042, Training Accuracy: 98.92%, Validation Loss: 0.0362, Validation Accuracy: 99.06%
Epoch [6/10], Training Loss: 0.9072, Training Accuracy: 99.71%, Validation Loss: 0.0595, Validation Accuracy: 98.12%
Epoch [7/10], Training Loss: 1.5747, Training Accuracy: 99.21%, Validation Loss: 0.0588, Validation Accuracy: 98.38%
Epoch [8/10], Training Loss: 1.9272, Training Accuracy: 99.04%, Validation Loss: 0.0437, Validation Accuracy: 98.50%
Epoch [9/10], Training Loss: 0.4479, Training Accuracy: 99.92%, Validation Loss: 0.0198, Validation Accuracy: 99.31%
Epoch [10/10], Training Loss: 0.3011, Training Accuracy: 99.96%, Validation Loss: 0.0371, Validation Accuracy: 98.75%
```

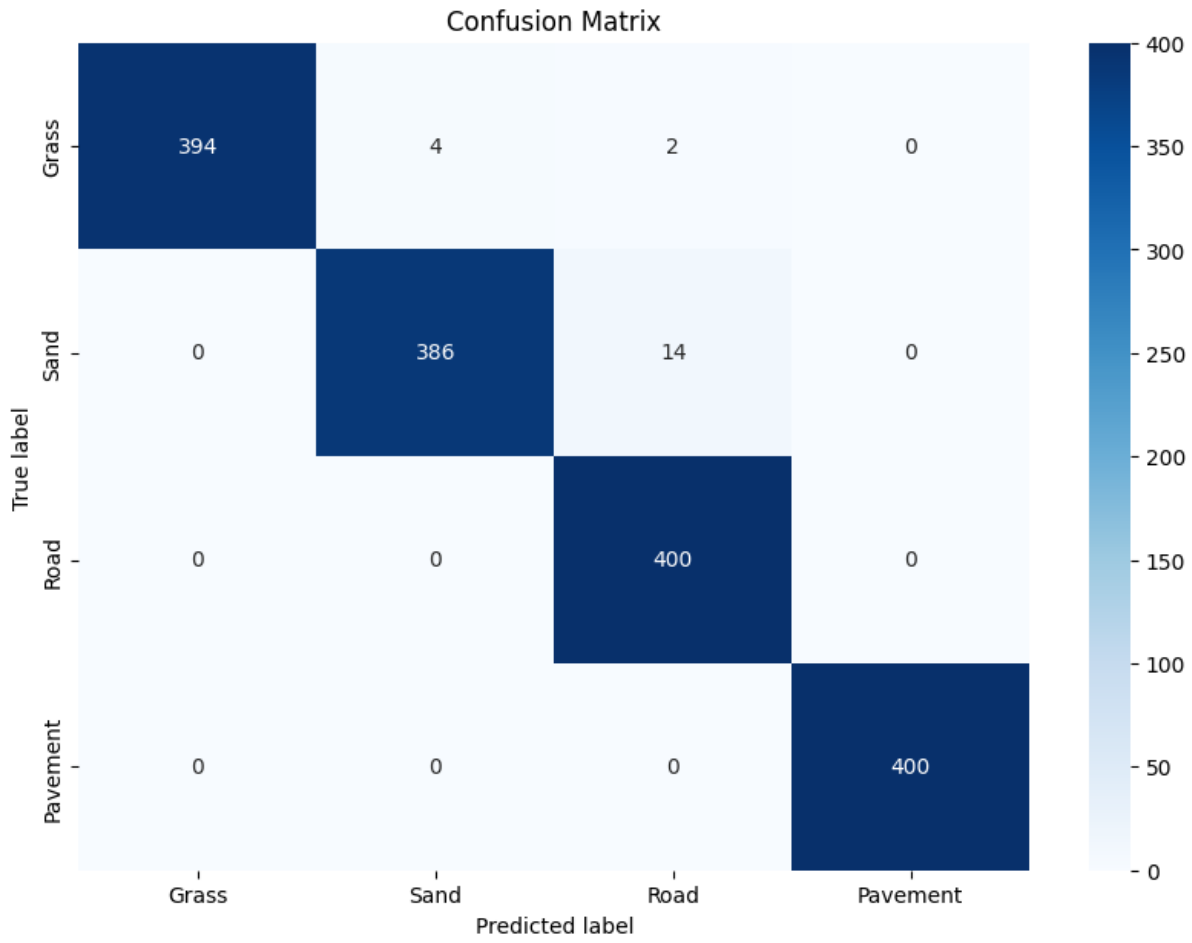
```python
# Best Validation Accuracy
print(f'Validation Accuracy: {best_val_accuracy:.2%}')
```

Python

```
Validation Accuracy: 99.31%
```

- Following are the Loss and Accuracy plots and the confusion matrix to determine how the model was able to predict for different classes:





- Then, the best model epoch was saved as a model.pth file. A code was written to evaluate the latency of the model and to evaluate it, in which the CNN model was defined, the best model file was imported to define the wights in the model, and then the model was provided with an input image and the results are as follows:

```
model_eval.py > ...
35  # Transformation for image processing
36  transform = transforms.Compose([
37      transforms.Resize((32, 32)),
38      transforms.ToTensor(),
39      transforms.Normalize(mean=[0.7720, 0.7452, 0.7508], std=[0.1130, 0.0996, 0.1095])
40  ])
41
42  # Image path
43  image_path = '/home/pi/Mini_Project/Dataset/Train/Pavement/image_20.jpg'
44
45  # Load the image and convert to tensor
46  image = Image.open(image_path)
47  input_tensor = transform(image).unsqueeze(0).to(device)  # Add batch dimension and move
48
49  # Evaluate the model
50  model.eval()
51  start_time = time.time()
52  output = F.softmax(model(input_tensor),dim=1)
53  end_time = time.time()
54
55  # Calculate the latency
56  latency = end_time - start_time
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

The latency of the model is: 0.008124828338623047 seconds
tensor([[2.7178e-03, 9.9715e-01, 1.3707e-04, 2.5857e-08]],
       grad_fn=<SoftmaxBackward0>)
Predicted class: Pavement
(mp) pi@raspberrypi:~ $
```

- Here, the latency is calculated using the time library in Python which comes out to be a staggering 8 ms.
- The output tensor showcases the probability of each class being detected in that image.
- As evident, the image provided was of the Pavement class, and the image was predicted correctly with overwhelmingly high probability.

5. Final Python Script for real-time Implementation

```
1   import serial
2   import time
3   import torch
4   import torch.nn as nn
5   import torch.nn.functional as F
6   from picamera2 import Picamera2, Preview
7   from torchvision import transforms
8   from PIL import Image
9
10  # Define the TerrainClassifier model
11  class TerrainClassifier(nn.Module):
12      def __init__(self):
13          super(TerrainClassifier, self).__init__()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Captured Image: test.jpg, Terrain Type: G
Terrain Type remains the same. Not sent to Arduino.
Captured Image: test.jpg, Terrain Type: G
Terrain Type remains the same. Not sent to Arduino.
Captured Image: test.jpg, Terrain Type: R
Captured Image: test.jpg, Terrain Type: R
Terrain Type 'R' sent to Arduino.
Captured Image: test.jpg, Terrain Type: R
Terrain Type remains the same. Not sent to Arduino.
Captured Image: test.jpg, Terrain Type: R
Terrain Type remains the same. Not sent to Arduino.
Captured Image: test.jpg, Terrain Type: R
Terrain Type remains the same. Not sent to Arduino.
Captured Image: test.jpg, Terrain Type: R
Terrain Type remains the same. Not sent to Arduino.
```

This final script is implementing a system for real-time terrain classification using a Raspberry Pi with a camera module and an Arduino. Here's a breakdown of its functionality:

- **Model Definition:** The script defines a convolutional neural network (CNN) model for terrain classification using PyTorch. The model architecture consists of two convolutional layers followed by max-pooling layers and two fully connected layers. The model is trained to classify images into four terrain classes: Grass (G), Pavement (P), Road (R), and Sand (S).
- **Model Loading:** The pre-trained model's state dictionary is loaded from a file (best_model_6.pth) to initialize the neural network. This file should contain the learned parameters of the trained model.
- **Camera Initialization:** The Raspberry Pi camera is initialized using the picamera2 library. It configures the camera settings for capturing images at two different resolutions: high resolution (1920x1080) and low resolution (640x480) for display purposes.
- **Serial Communication:** A serial connection is established between the Raspberry Pi and the Arduino using the serial library. The script opens a serial port (/dev/ttyACM0) with a baud rate of 9600. This connection allows communication between the Raspberry Pi and Arduino for speed control via the Arduino.
- **Real-time Image Processing:** The script captures images using the Raspberry Pi camera and saves them as test.jpg. These images are then resized to 32x32 pixels and pre-processed (converted to tensors and normalized) using PyTorch transforms.
- **Inference:** The pre-processed image tensors are fed into the trained CNN model for inference. The model predicts the terrain class probabilities using softmax activation and selects the class with the highest probability as the predicted terrain type.
- **Terrain Classification:** The predicted terrain type is mapped to one of the four terrain classes (Grass, Pavement, Road, Sand) based on the model's output.
- **Moving Window Method for enhanced accuracy:** If the predicted terrain changes for two consecutive readings, only then does the script send a terrain message to the Arduino via serial communication. This ensures that if an occasional misclassification occurs, the performance of the car is not affected. The terrain type is sent in uppercase format.
- **Output Display and Control:** The script prints the captured image filename (test.jpg) and the corresponding terrain type to the console. It also sends terrain messages to the Arduino to control the speed of the vehicle based on the detected terrain type.
- **Interrupt Handling**: The script handles keyboard interrupts (Ctrl+C) gracefully by closing the serial communication port (ser.close()) before exiting.

Overall, this script demonstrates a real-time system for terrain classification using deep learning on a Raspberry Pi and Arduino integration for controlling speed based on the detected terrain type.

Applications

- **In Autonomous Vehicles:** The project's capability to recognize and classify different terrains can significantly enhance the safety and efficiency of autonomous vehicles. By adjusting speed and changing car modes according to various terrain types such as snow, mud, grass, sand, or paved roads, these vehicles can offer a smoother and safer driving experience for passengers. This adaptability ensures optimal vehicle performance while navigating through diverse environments.

- **In Military Systems:** The ability to identify and classify different terrains is crucial for military operations. The project's technology can assist in the planning and execution of missions by providing critical information about the terrain soldiers may encounter. Understanding the terrain in advance can improve tactical decisions and logistics planning. Additionally, this technology can be used in unmanned military vehicles, allowing them to navigate complex terrains autonomously, thereby reducing risk to human personnel.

- **In Space Rovers:** Terrain recognition capabilities have direct applications in the development of space rovers, enabling them to traverse unknown planets and moons safely and efficiently. As rovers collect scientific data and carry out missions in challenging environments, the ability to adapt to different types of terrain can improve the rover's effectiveness and longevity in harsh extraterrestrial conditions.

- **In Agricultural Technology:** The project's ability to recognize and classify terrain types can also benefit precision agriculture. By identifying different soil and land types, agricultural vehicles can adjust their operations, such as planting or harvesting, to optimize crop yield and resource usage.

- **In Search and Rescue Operations:** Terrain recognition technology can assist search and rescue teams in navigating diverse and challenging environments. By providing information about the terrain, teams can plan their approach more effectively and safely to reach individuals in need.

- **In Geology and Environmental Monitoring:** The ability to classify different types of terrain can help geologists and environmental scientists study changes in landscapes and monitor geological events such as landslides or erosion. This technology can provide valuable insights into natural processes and aid in environmental protection efforts.

These applications demonstrate the versatility and potential impact of the project's technology across multiple domains. By providing autonomous systems with the ability to recognize and respond to different terrains, this project opens up new possibilities for innovation and efficiency in a range of industries.

Conclusion

The TerrainRover AI is a low-cost prototype now capable of detecting terrain types with 99.31% accuracy with just an 8ms latency in real-time and implementing speed control for various terrain types along with the ability to be controlled manually via Bluetooth on a smartphone app.

Finance Details

No	Component	Cost
1	Arduino	580
2	Raspberry Pi Model 4B	8000(Taken from Faculty)
3	L298N	200
4	Bluetooth Module	210
5	Connecting Wires	20
6	Power Bank	1200
7	18650 Batteries	188
8	Screws and Bolts	632
9	Battery Charger	200
10	Battery Holder	120
11	BO Motors	310
12	Wheels 65mm	182
13	PiCamera	380
14	Camera Mount	200
15	Insulation tape and tester	136
16	L-Clamp	80
	Total	12638

References

1. Deepterramechanics: Terrain classification and slip estimation for ground robots via deep learning
R Gonzalez, K Iagnemma - arXiv preprint arXiv:1806.07379, 2018 - arxiv.org

2. Improving robot mobility by combining downward-looking and frontal cameras
R Gonzalez, A Rituerto, JJ Guerrero - Robotics, 2016 - mdpi.com

3. Optimum pipeline for visual terrain classification using improved bag of visual words and fusion methods
H Wu, B Liu, W Su, Z Chen, W Zhang, X Ren, J Sun
Journal of Sensors, 2017 • hindawi.com

4. Terrain Classification for Off-Road Driving
K Shen, M Kelly, S Le Cleac'h
Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Project Rep, 2017 • cs229.stanford.edu